Comp 310
Computer Systems and Organization

Lecture #19

File Systems

(File & Directory Basics – Part 1)

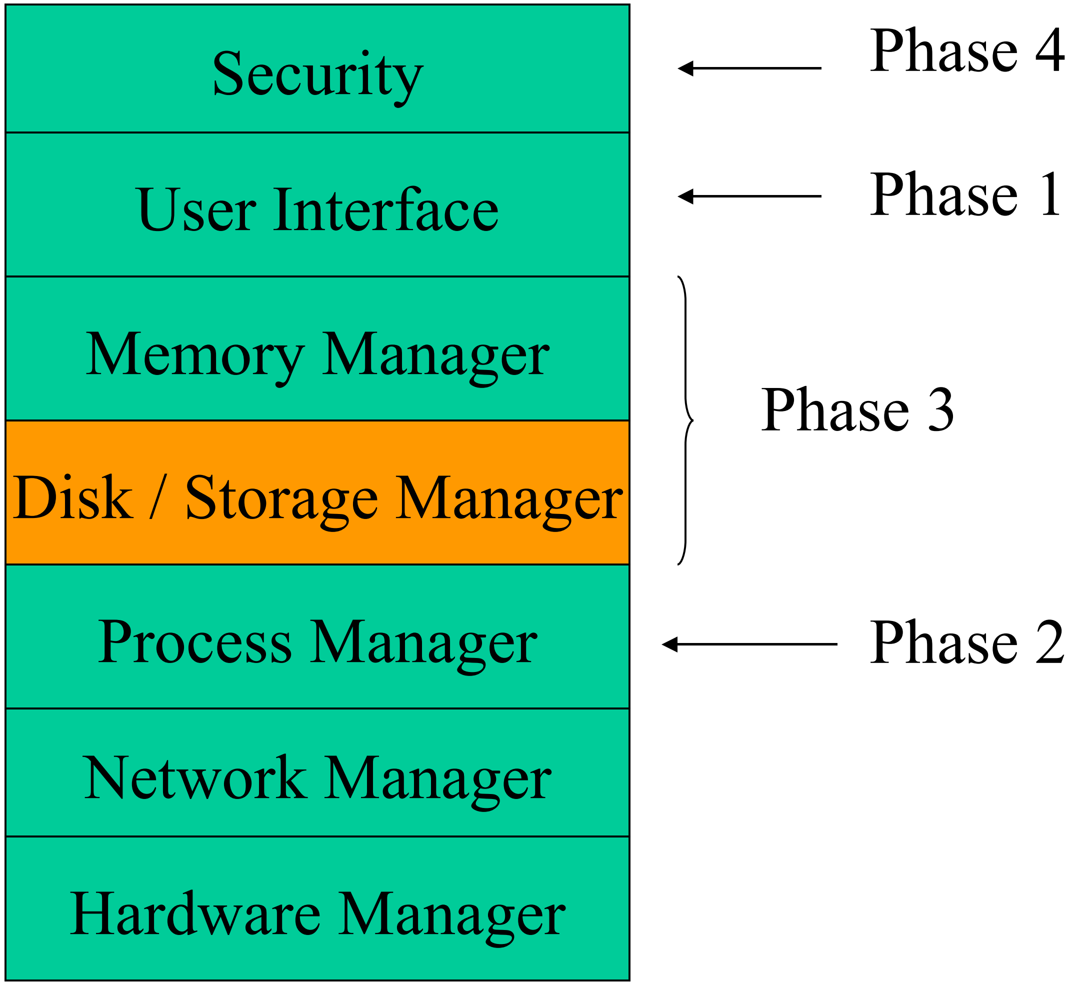
Prof. Joseph Vybihal

Announcements

- Final Exam
  - Dec 9, 2PM

Basic OS Architecture

(Course Table of Contents)



Part 1

About Files

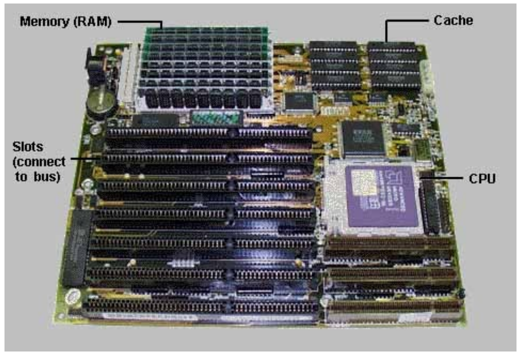
A File System

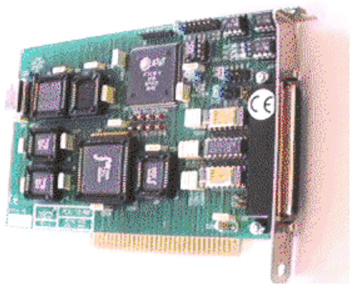
- Definition:
  - A method by which the OS imposes a technique where by it understands the meaning of files, storage, retrieval and access.
- Two views:
  - User view
    - (i.e. window's folders, Unix's directories)
  - Actual Implementation
    - (data structures, device controllers)

Programs and more programs

OS in RAM

Icons, folders & shortcuts



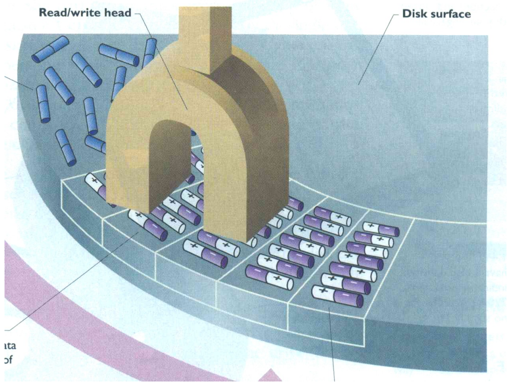


Driver on card

The OS

Firmware on HDD

Actual medium



What makes up a file?

(logically)

File Attributes

- A symbolic *Name*
- A unique integer *Identifier*
- Its *Type*
- An *Address* on disk
- Its *Size* in bytes
- The *Security* privileges assigned to the file
- Who is the *Owner* of the file
- Dates: *Creation, Modification, Time*

Where should we store this info?

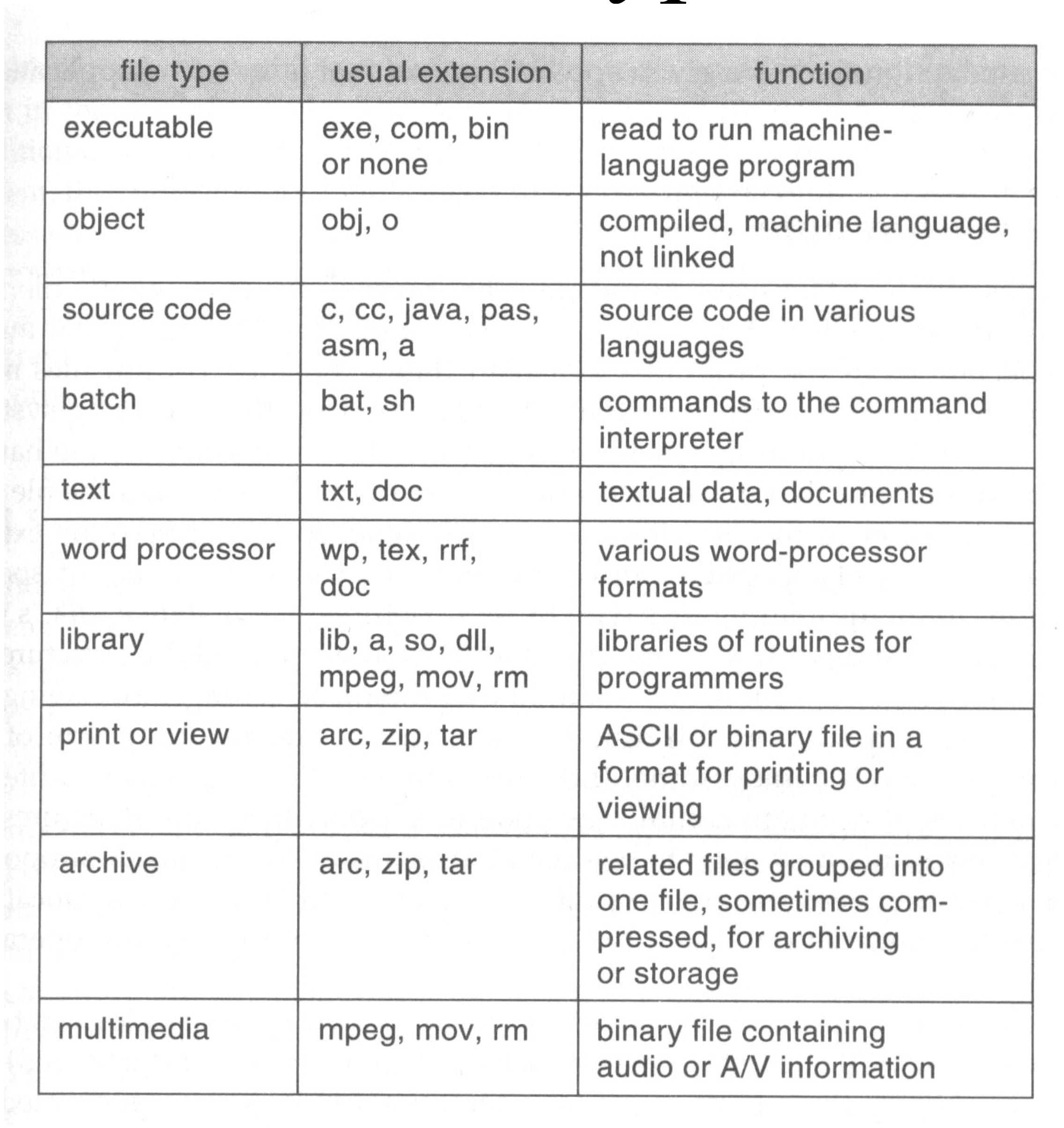
Others?

File Operations

- Create a file
  - Text mode
  - Binary mode
- Write to a file (by mode)
- Reading from a file (by mode)
- Repositioning within a file
  - Sequential
  - Random
  - Reverse order
- Deleting a file
- Appending to a file
- Truncating a file
  - Delete file
  - But keep attributes
  - Write to file

Are these OS or Language managed?

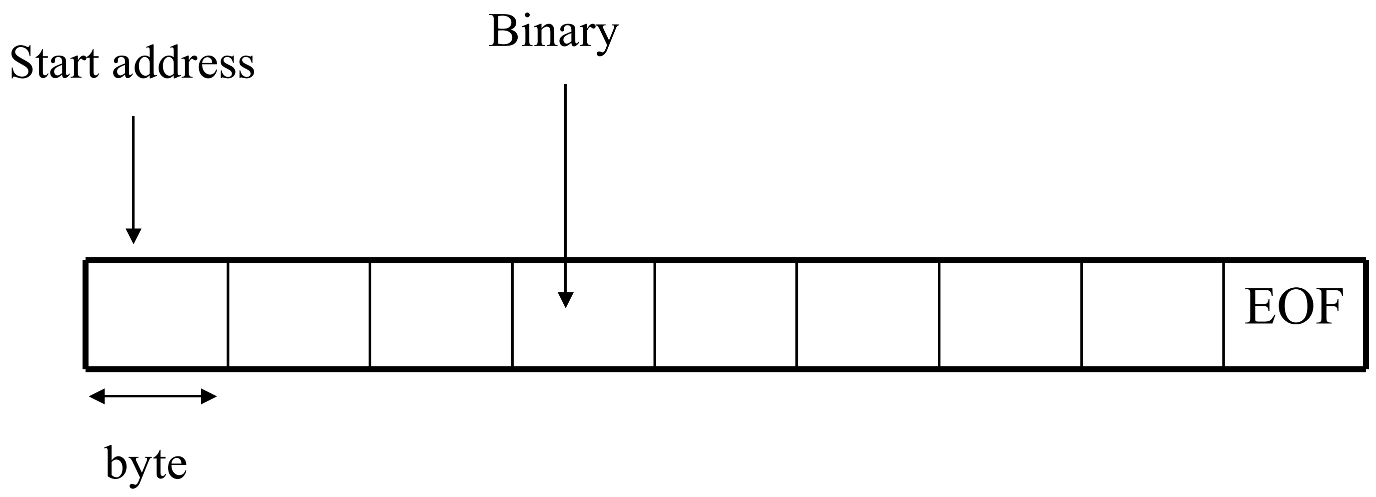
File Types



file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

Are they or should they be formatted differently?

Basic File Structure



Byte addressable

Should the OS provide for more complex file structures?

Example

Dear Mom,
  Thanks for the money! You are the best.
Love Joe.

D	e	a	r		M	o	m	,	cr	lf	tab	T	h
a	n	k	s		f	o	r		t	h	e		m
o	n	e	y	!		Y	o	u		a	r	e	
t	h	e		b	e	s	t	.	cr	lf	L	o	v
e		J	o	e	.	EOF	▨	▨	▨	▨	▨	▨	▨

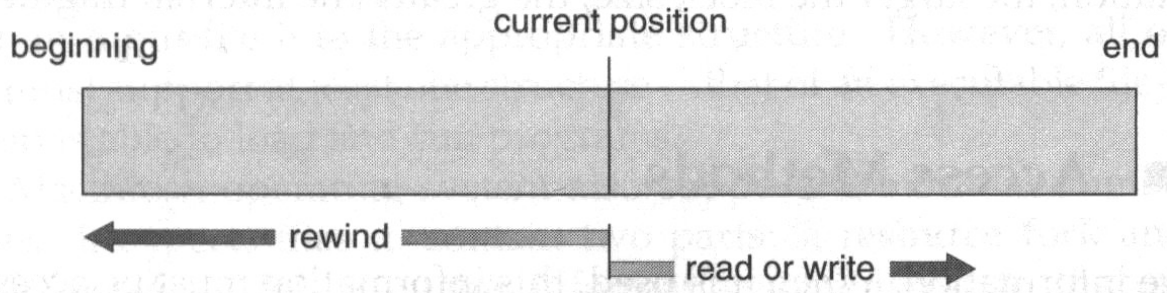
Question

- How could we implement this physically on a disk? What would the OS need to do?

  (think of this in C)

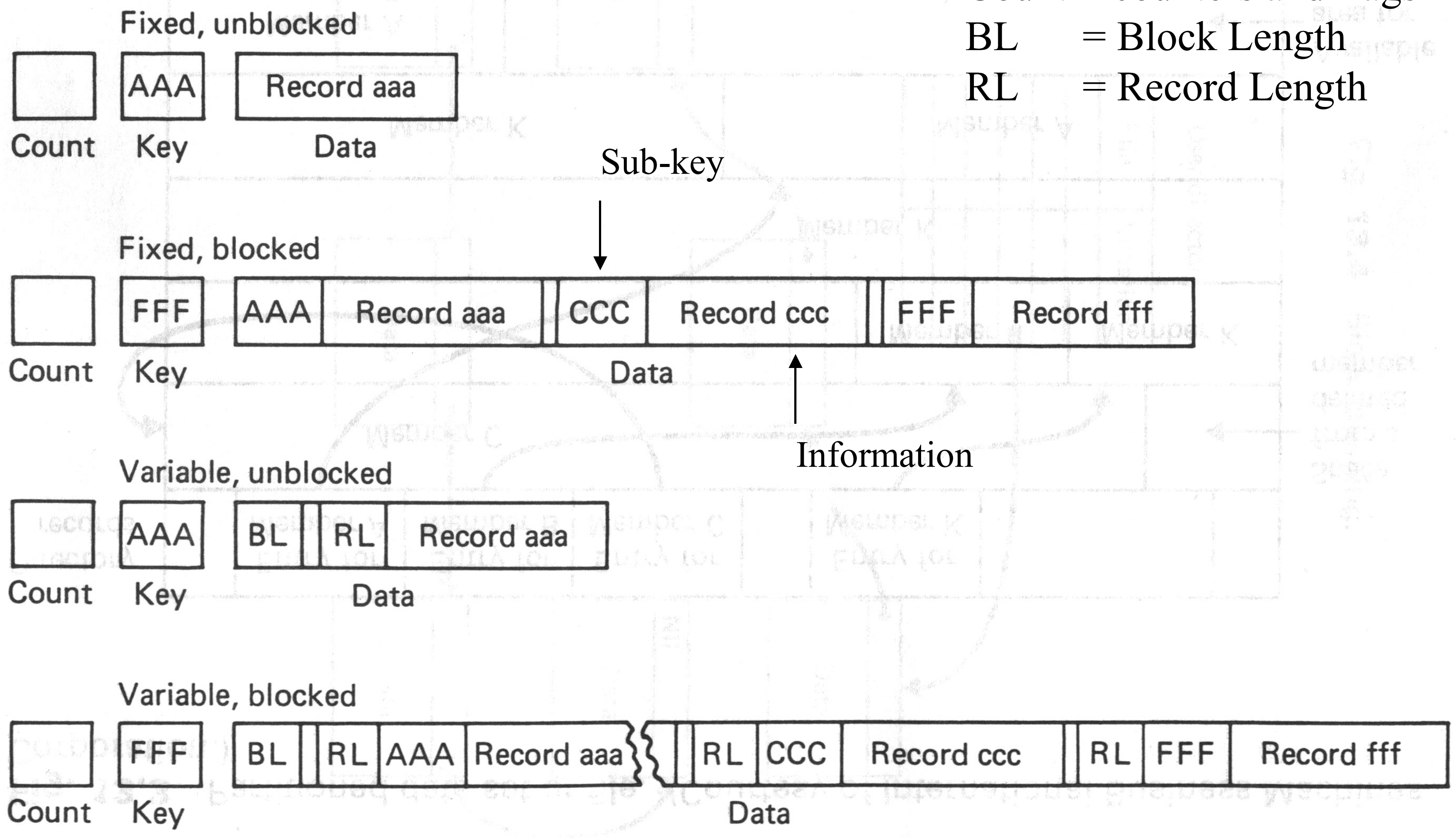
Sequential Access

sequential access	implementation for direct access
reset	*cp* = *0*;
read next	*read cp*; *cp* = *cp*+1;
write next	*write cp*; *cp* = *cp*+1;



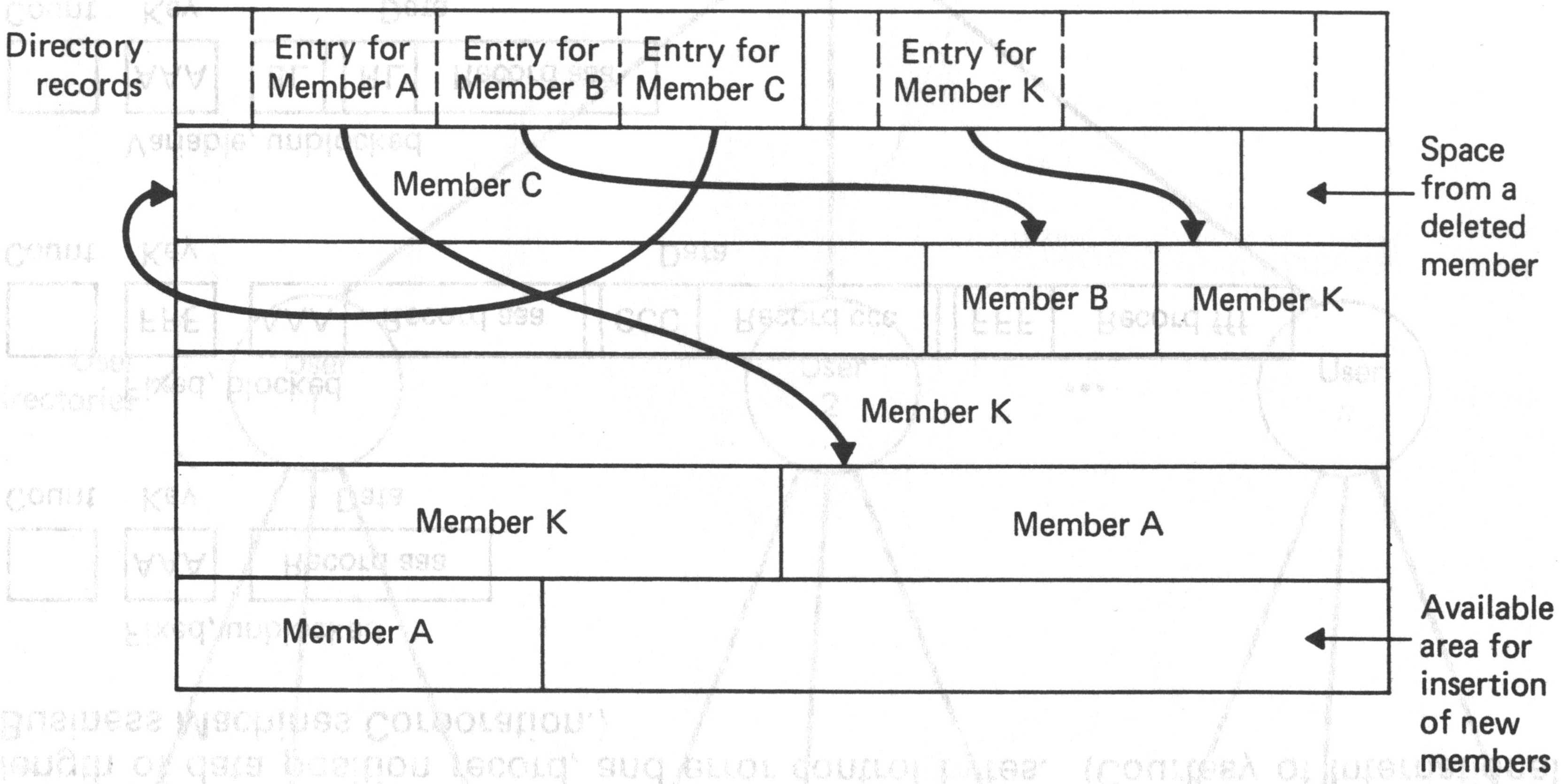
AS400 File Structures

Count = counters and flags
BL = Block Length
RL = Record Length



Indexed Sequential Files ISAM Files

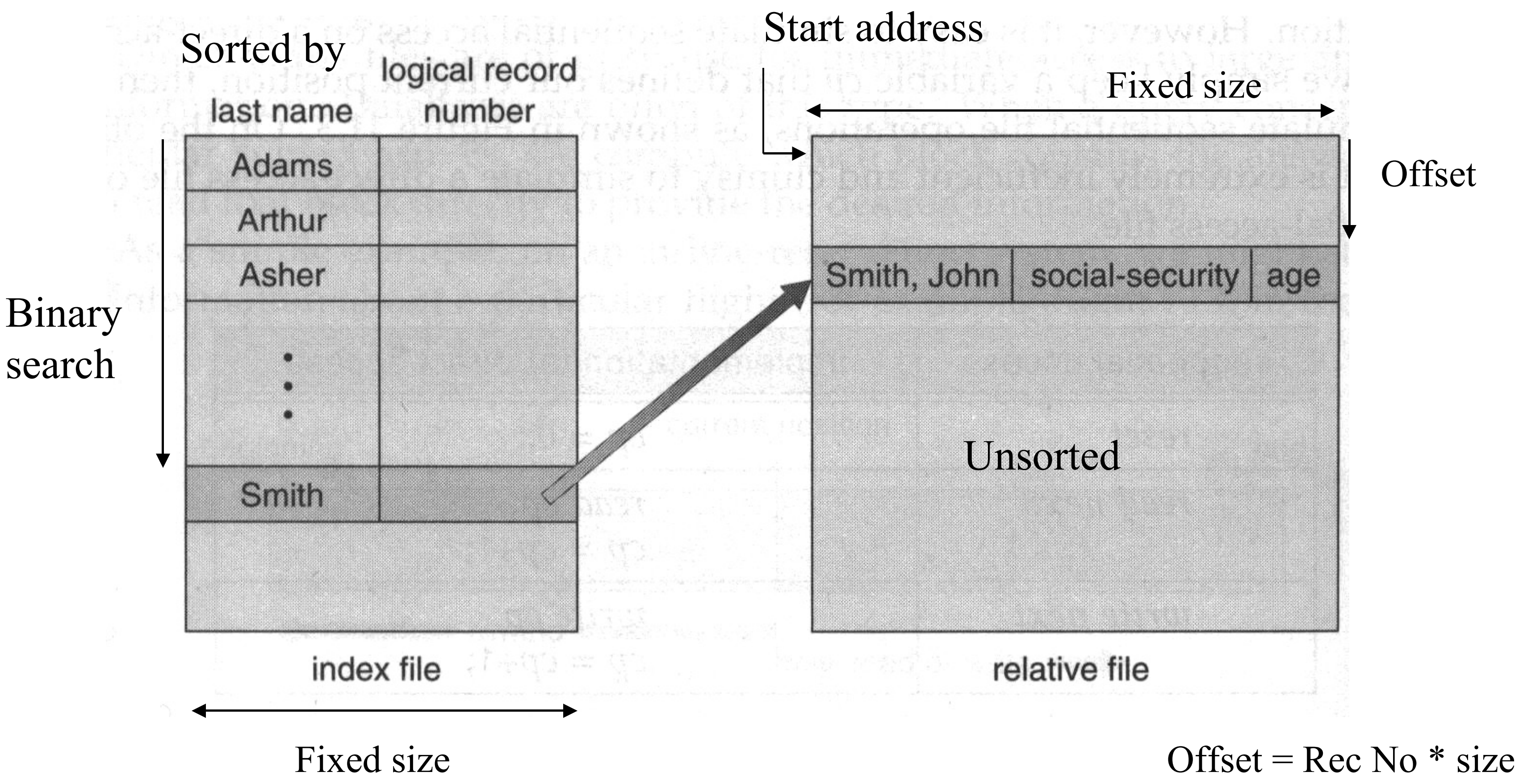
AS400 File Structures



<u>Partitioned File</u> (a set of Sequential Files with a directory header)

Software Supported File Structures

Database Example



How do we do binary search?

File Access Methods

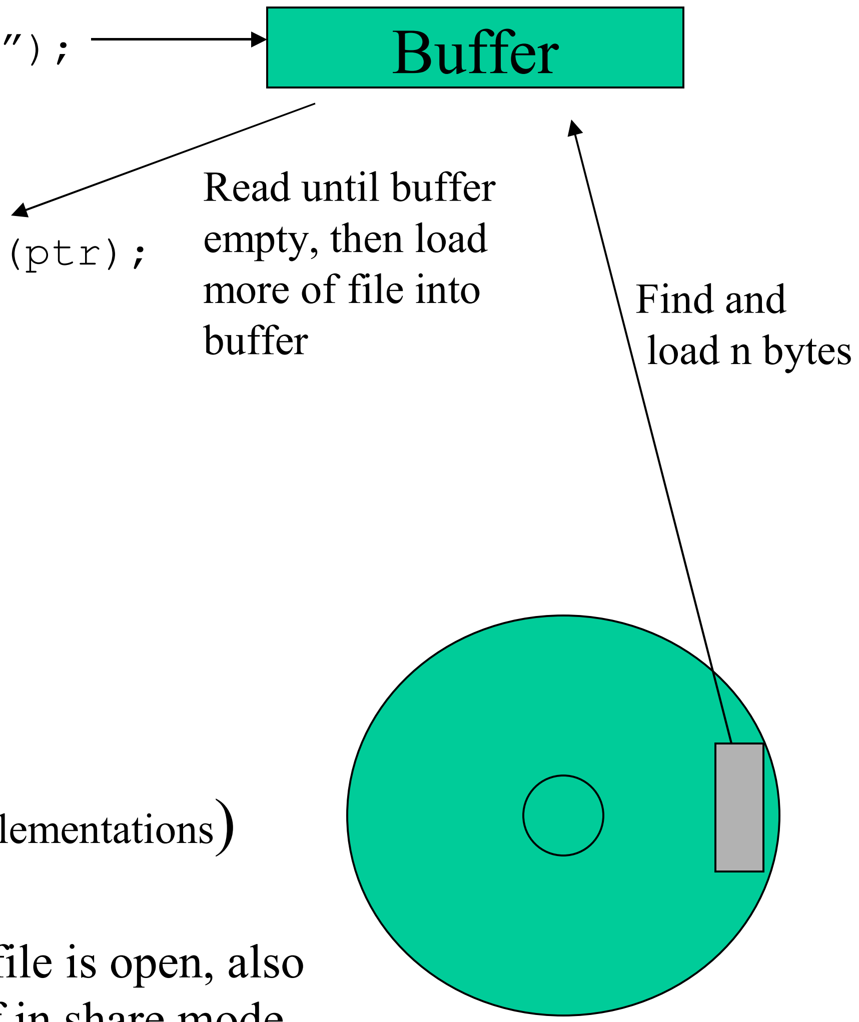
- In all cases:
  - File = Open(type, style)
    - File is a pointer to the buffer in memory
    - Type = text, binary, indexed, partitioned, …
    - Style = read, write, append, …
  - Close(File)
    - Flushes buffer and frees buffer
- Sequential
  - Fscanf: read one byte, pointer moves to next byte
  - Fprintf: write one byte, pointer moves to next byte
- Direct
  - Fread: n bytes read from disk
  - Fwrite: n byte write to disk
  - Fseek: start position, offset in bytes
    - Start position: beginning, end, current position
- Indexed
  - Iseek: Key
  - Iread & Iwrite: one record

OS Buffers

```
FILE *ptr = fopen("abc.txt","rt");
char x[100];
int I;

for(I=0;I<100;I++) x[I] = fgetc(ptr);

fclose(ptr);
```



Delete buffer

The file access commands must interface with OS not with the files directly! (in buffer implementations)

Record file is open, also record if in share mode …

Low-level File Access in C

```
#define   CMASK     0377 /* for making char's > 0 */
#define   BUFSIZE   512

getchar() /* buffered version */
{
     static char    buf[BUFSIZE];
     static char    *bufp = buf;
     static int     n = 0;

     if (n == 0) {  /* buffer is empty */
          n = read(0, buf, BUFSIZE);
          bufp = buf;
     }
     return((--n >= 0) ? *bufp++ & CMASK : EOF);
}
```

Octal: 377 = 011111111 (9 bin char) Sign forced to 0

Numeric file descriptor (max 15-30) PCB file ptr array

```
get(fd, pos, buf, n) /* read n bytes from position pos */
int fd, n;
long pos;
char *buf;
{
     lseek(fd, pos, 0);   /* get to pos */
     return(read(fd, buf, n));
}
```

Number of bytes to address

Start of file

```
#define NULL 0
#define BUFSIZE 512
#define PMODE 0644 /* RW for owner, R for group, others */

main(argc, argv)     /* cp: copy f1 to f2 */
int argc;
char *argv[];
{
     int  f1, f2, n;
     char buf[BUFSIZE];

     if (argc != 3)
          error("Usage: cp from to", NULL);
     if ((f1 = open(argv[1], 0)) == -1)
          error("cp: can't open %s", argv[1]);
     if ((f2 = creat(argv[2], PMODE)) == -1)
          error("cp: can't create %s", argv[2]);

     while ((n = read(f1, buf, BUFSIZE)) > 0)
          if (write(f2, buf, n) != n)
               error("cp: write error", NULL);
     exit(0);
}

error(s1, s2)  /* print error message and die */
char *s1, *s2;
{
     printf(s1, s2);
     printf("\n");
     exit(1);
}
```

Open ➔ open read
Create ➔ open write
Read
Write
Exit ➔ closes all files

stdio.h

```
#define   _BUFSIZE  512
#define   _NFILE    20    /* #files that can be handled */

typedef struct _iobuf {
     char *_ptr;      /* next character position */
     int  _cnt;       /* number of characters left */
     char *_base;     /* location of buffer */
     int  _flag;      /* mode of file access */
     int  _fd;        /* file descriptor */
} FILE;
extern FILE _iob[_NFILE];

#define   stdin     (&_iob[0])
#define   stdout    (&_iob[1])
#define   stderr    (&_iob[2])

#define   _READ     01    /* file open for reading */
#define   _WRITE    02    /* file open for writing */
#define   _UNBUF    04    /* file is unbuffered */
#define   _BIGBUF   010   /* big buffer allocated */
#define   _EOF 020   /* EOF has occurred on this file */
#define   _ERR 040   /* error has occurred on this file */
#define   NULL  0
#define   EOF   (-1)
#define   getc(p)   (--(p)->_cnt >= 0 \
               ? *(p)->_ptr++ & 0377 : _fillbuf(p))
#define   getchar() getc(stdin)

#define   putc(x,p) (--(p)->_cnt >= 0 \
               ? *(p)->_ptr++ = (x) : _flushbuf((x),p))
#define   putchar(x)     putc(x,stdout)
```

```
#include  <stdio.h>
#define   PMODE     0644 /* R/W for owner; R for others */

FILE *fopen(name, mode)  /* open file, return file ptr */
register char *name, *mode;
{
     register int fd;
     register FILE *fp;

     if (*mode != 'r' && *mode != 'w' && *mode != 'a') {
          fprintf(stderr, "illegal mode %s opening %s\n",
               mode, name);
          exit(1);
     }
     for (fp = _iob; fp < _iob + _NFILE; fp++)
          if ((fp->_flag & (_READ | _WRITE)) == 0)
               break;    /* found free slot */
     if (fp >= _iob + _NFILE) /* no free slots */
          return(NULL);

     if (*mode == 'w')   /* access file */
          fd = creat(name, PMODE);
     else if (*mode == 'a') {
          if ((fd = open(name, 1)) == -1)
               fd = creat(name, PMODE);
          lseek(fd, 0L, 2);
     } else
          fd = open(name, 0);
     if (fd == -1)  /* couldn't access name */
          return(NULL);

     fp->_fd = fd;
     fp->_cnt = 0;
     fp->_base = NULL;
     fp->_flag &= ~(_READ | _WRITE);
     fp->_flag |= (*mode == 'r') ? _READ : _WRITE;
     return(fp);
}
```

```
directory(name)      /* fsize for all files in name */
char *name;
{
     struct direct dirbuf;
     char *nbp, *nep;
     int i, fd;

     nbp = name + strlen(name);
     *nbp++ = '/';   /* add slash to directory name */
     if (nbp+DIRSIZ+2 >= name+BUFSIZE)   /* name too long */
          return;
     if ((fd = open(name, 0)) == -1)
          return;
     while (read(fd, (char *)&dirbuf, sizeof(dirbuf))>0) {
          if (dirbuf.d_ino == 0)    /* slot not in use */
               continue;
          if (strcmp(dirbuf.d_name, ".") == 0
            || strcmp(dirbuf.d_name, "..") == 0)
               continue; /* skip self and parent */
          for (i=0, nep=nbp; i < DIRSIZ; i++)
               *nep++ = dirbuf.d_name[i];
          *nep++ = '\0';
          fsize(name);
     }
     close(fd);
     *--nbp = '\0'; /* restore name */
}
```

```
#define   DIRSIZ    14    /* max length of file name */

struct direct   /* structure of directory entry */
{
     ino_t d_ino;    /* inode number */
     char d_name[DIRSIZ];      /* file name */
};
```

Part 2

File Systems

File Allocation Table (FAT)

- A data structure on the storage device needs to record the information for a file:
  - Name, size, owner, security, location, type, dates

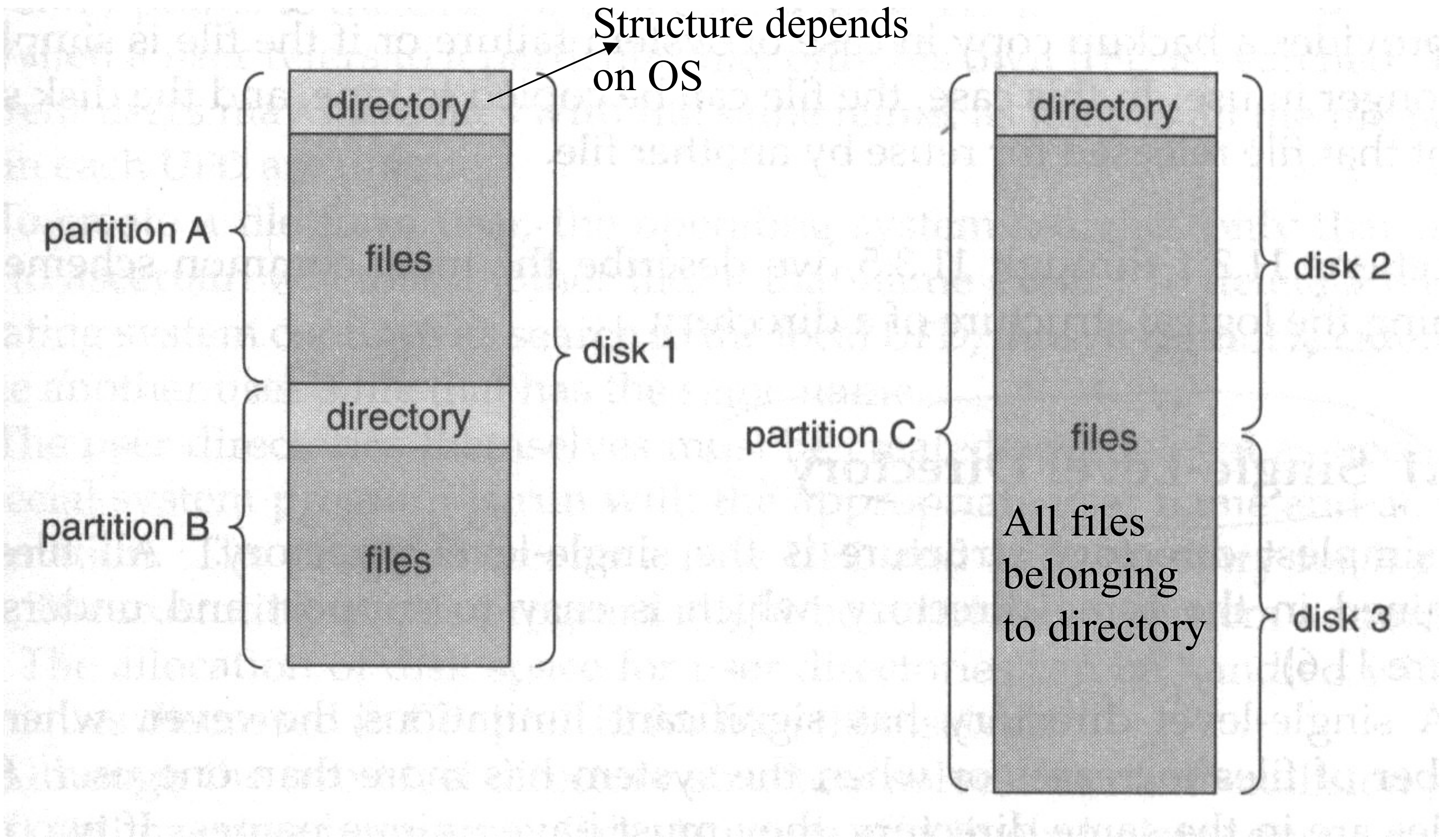
Name	Size	Owner	Security	Address	Type	Date
abc	100	Jack	Priv	F01	txt	…

Depends on addressing method of device

Space limitations?

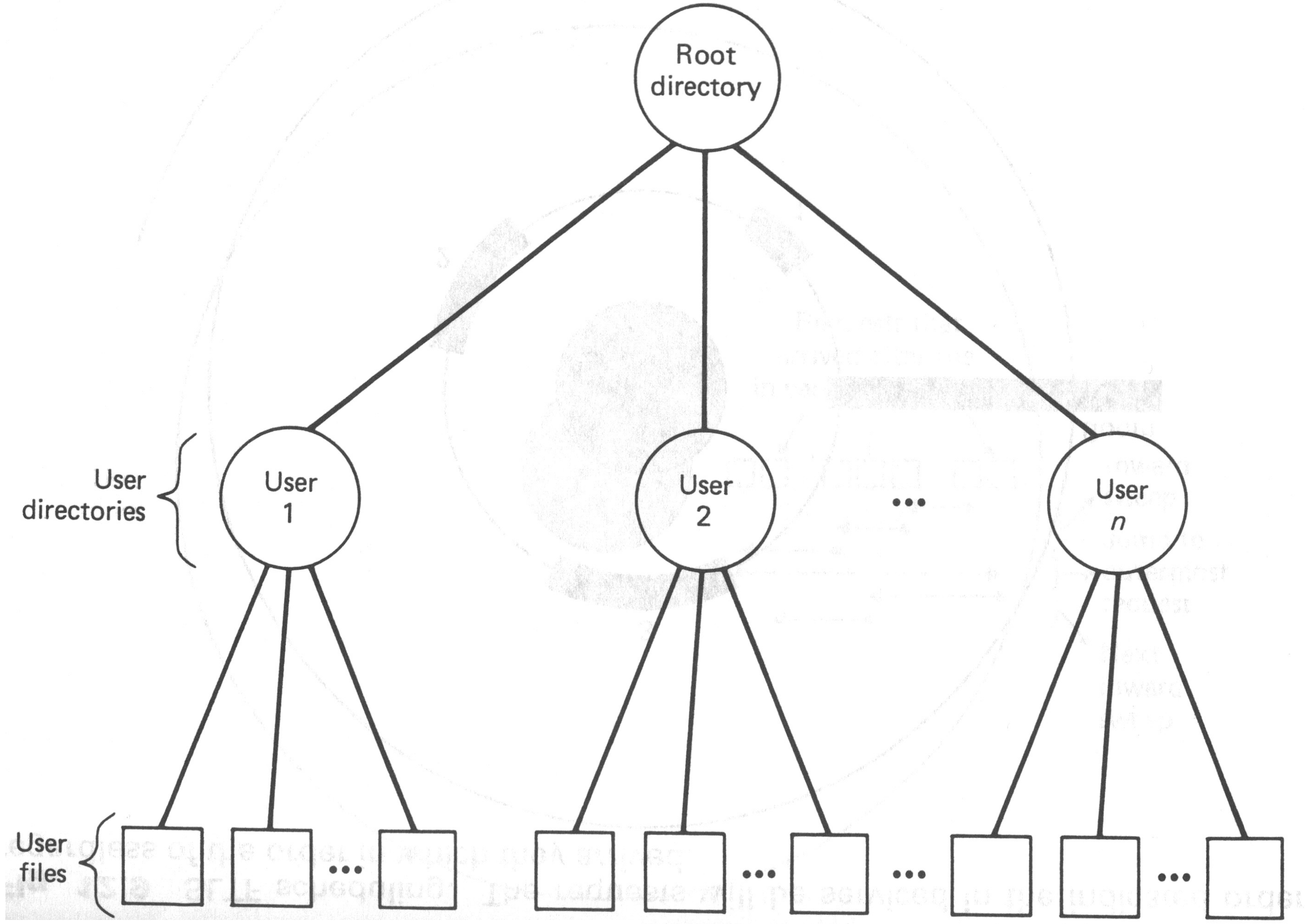
Typical Organization

Partitions



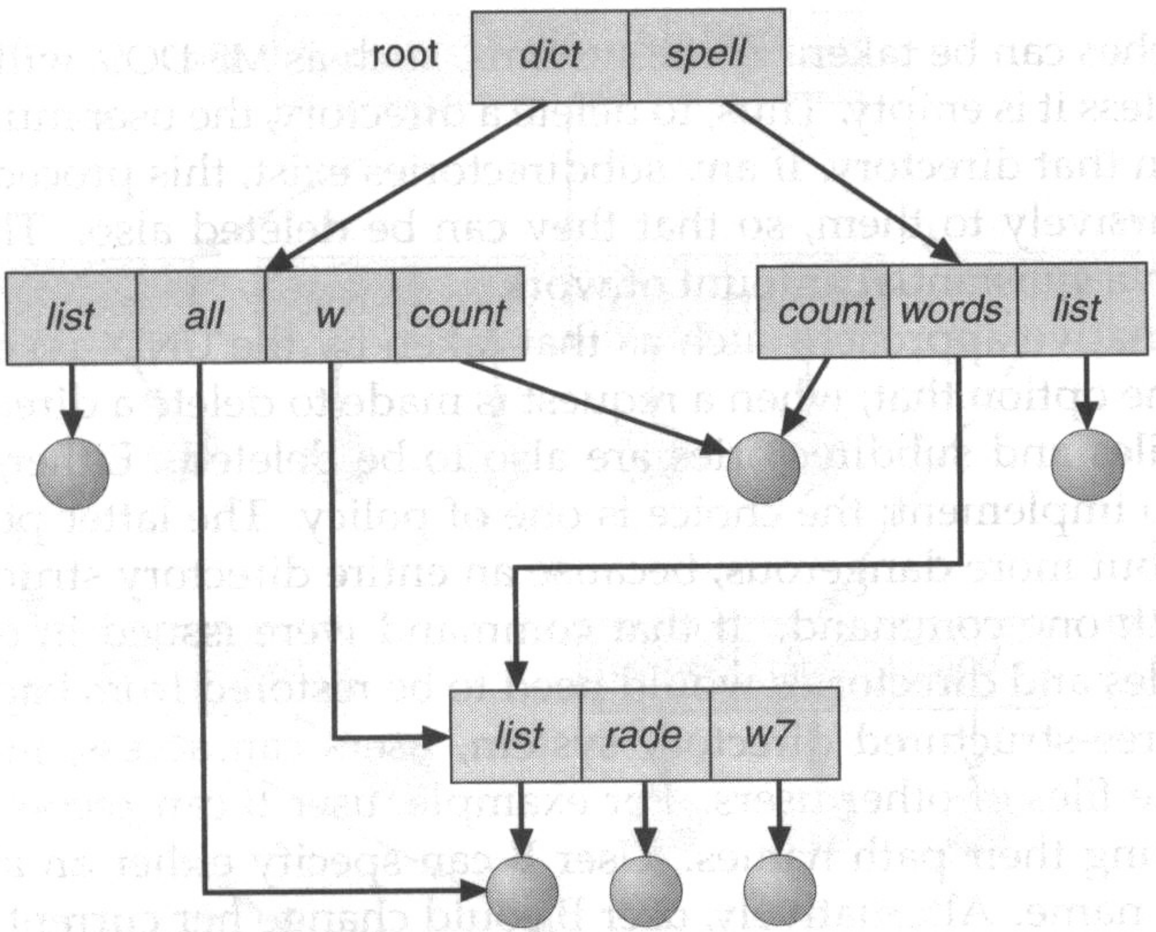
FAT Structures

Multi-Layered



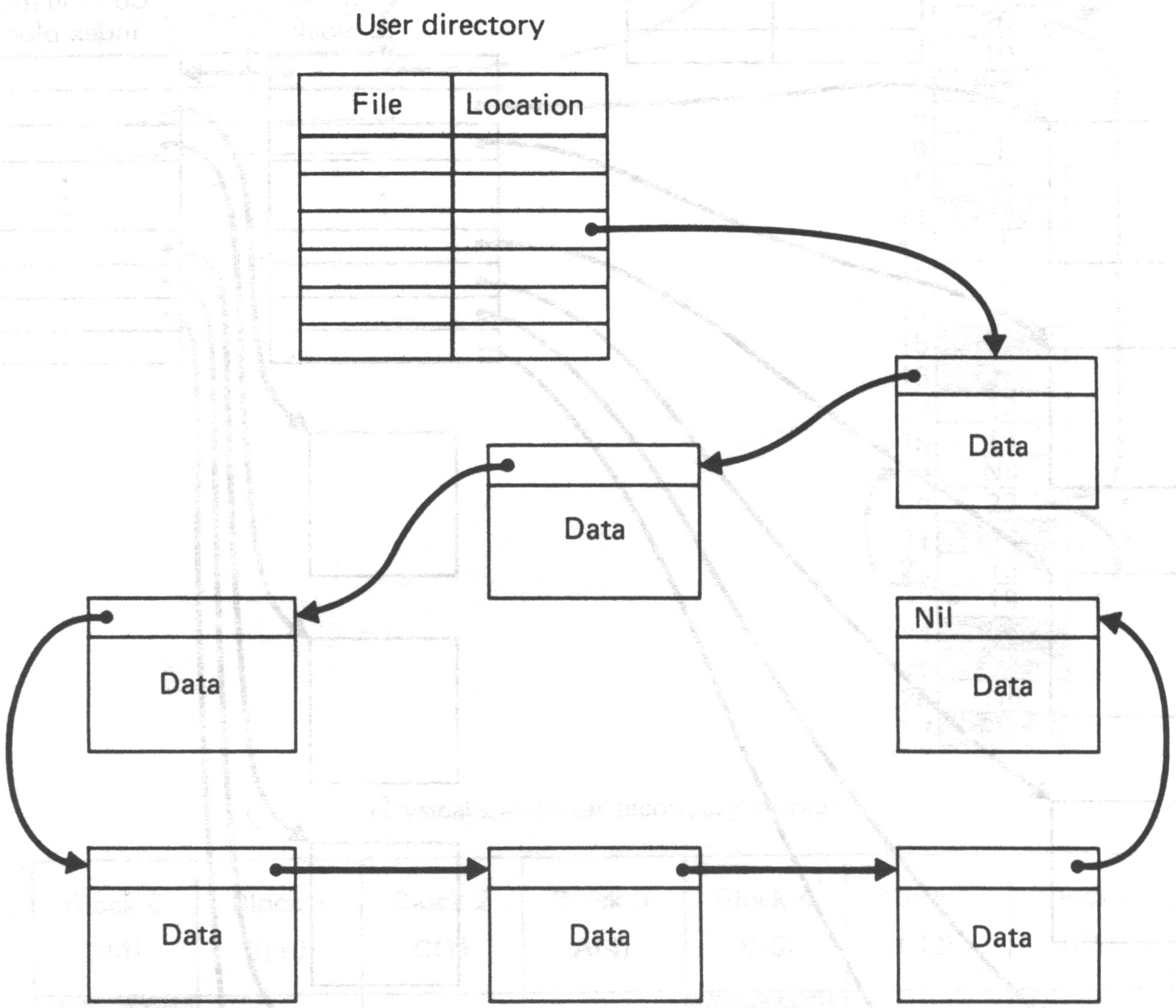
Acyclic Graph Directories

Permits shared files and directories

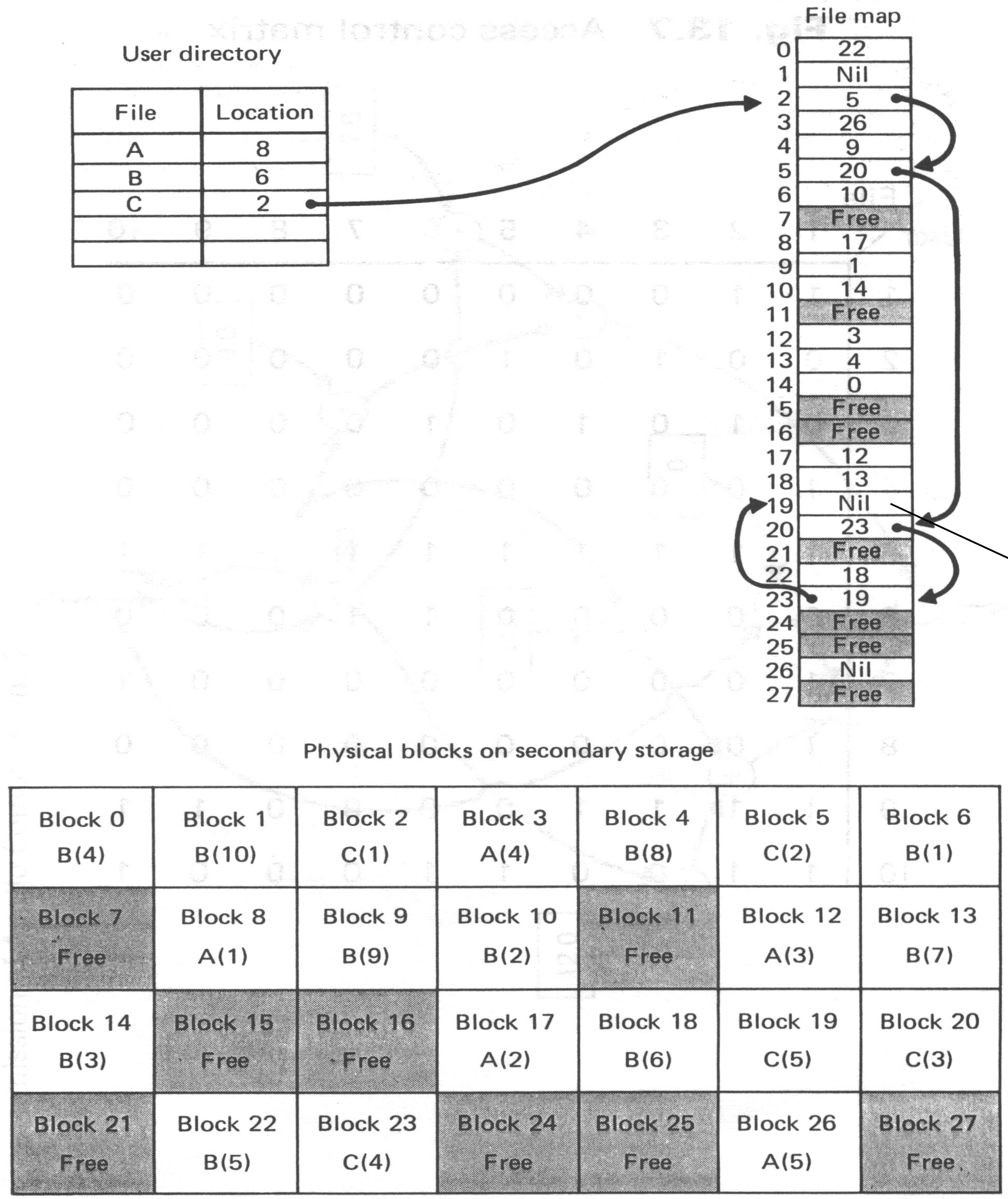


File Block Chaining

When your file cannot fit in a contiguous space on the medium



Block-Oriented File Mapping



User directory

File	Location
A	8
B	6
C	2

File map

0	22
1	Nil
2	5
3	26
4	9
5	20
6	10
7	Free
8	17
9	1
10	14
11	Free
12	3
13	4
14	0
15	Free
16	Free
17	12
18	13
19	Nil
20	23
21	Free
22	18
23	19
24	Free
25	Free
26	Nil
27	Free

Physical blocks on secondary storage

Block 0 B(4)	Block 1 B(10)	Block 2 C(1)	Block 3 A(4)	Block 4 B(8)	Block 5 C(2)	Block 6 B(1)
Block 7 Free	Block 8 A(1)	Block 9 B(9)	Block 10 B(2)	Block 11 Free	Block 12 A(3)	Block 13 B(7)
Block 14 B(3)	Block 15 Free	Block 16 Free	Block 17 A(2)	Block 18 B(6)	Block 19 C(5)	Block 20 C(3)
Block 21 Free	Block 22 B(5)	Block 23 C(4)	Block 24 Free	Block 25 Free	Block 26 A(5)	Block 27 Free

This has data and is the end

Question

- In each of the file mapping techniques, how could we implement them in C?
  - Data structures?
  - Algorithms?

Part 3

At Home

Things to try out

1. Use a program like Norton Disk Doctor to look at the byte structure of your files. Then modify them, even try to mess up some files.
   - This is a relatively safe operation if you select safe files like text or word files.